Computer Science Department

TECHNICAL REPORT

TIGHT AREA BOUNDS AND PROVABLY GOOD AT^2 BOUNDS FOR SORTING CIRCUITS

By Alan Siegel
October 1985

Technical Report #122

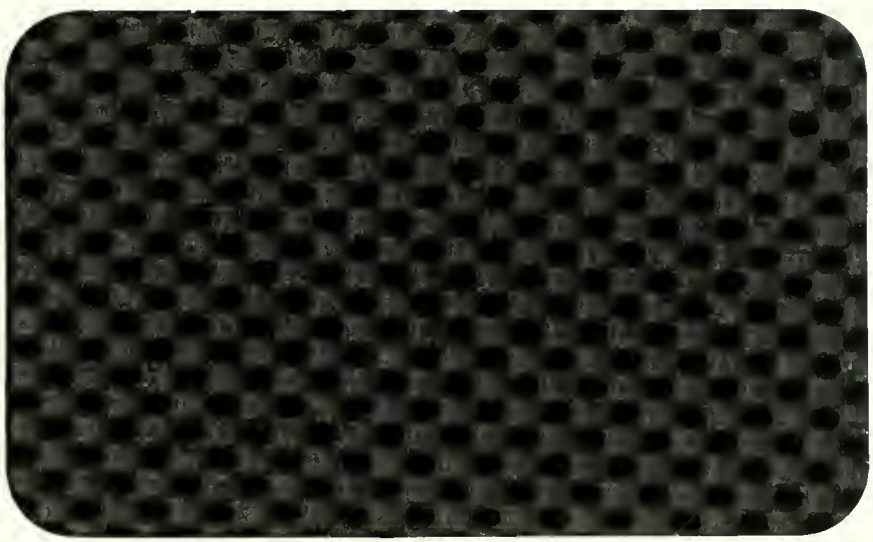
REVISED

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012

NYU COMPSCI TR-122 C.1
Siegel, Alan
Tight area bounds and
provably good AT{sup}2
bounds... Revised

CS-TR-122-Rev



TIGHT AREA BOUNDS AND PROVABLY GOOD AT^2 BOUNDS FOR SORTING CIRCUITS

By Alan Siegel
October 1985

Technical Report #122

REVISED

Tight Area Bounds and Provably Good AT^2 Bounds for Sorting Circuits

Alan Siegel
Courant Institute

Abstract

This paper gives tight upper and lower bounds for the minimum area required to sort n k-bit numbers in a digital medium, when the inputs can be replicated up to $r \leq n$ times. We also give provably good AT^2 bounds for VLSI sorting circuits that read their inputs once. Our lower bounds result from a coherent theory that captures the intrinsic complexity in both A and AT^2 measures for sorting circuits. Among other results, we prove:

(1) The minimum area $A(n,k,r)$ required to sort n k-bit numbers, which can be read r times, satisfies

$$A(n,k,r) = \Theta \begin{cases} \log n + 2^k \log \frac{n}{r2^{k-1}}, & 2^k \leq n/r, \\ \log n + \frac{n}{r} \log \frac{r2^{k+1}}{n}, & n/r \leq 2^k \leq n^{O(1)}. \end{cases}$$

(2) A VLSI circuit that sorts n k-bit numbers, for $\log n \leq k \leq 2\log n$, and that reads its inputs exactly once satisfies

$$AT^2 = \Omega(n^2 \log^2 \frac{2^{k+1}}{n}).$$

1. Introduction

"Minimum Storage Sorting Networks" [S1] analyzes the problem of how to sort n k-bit numbers in a minimum area VLSI circuit. Lower bounds for the circuit area are established, and sorting circuits are described, which require an area that is within a constant factor of optimal. These results are based on the fact that the inputs are read only once. We now analyze the problem when the inputs are read $r \leq n$ times. This problem is clearly more difficult, and its solution requires new techniques to measure how much information a sorting circuit must be able to retain.

In [S1], bounds are established by showing that a circuit, at times, has to store a certain amount of information, lest it fail to sort correctly. Also, a Testing Lemma is used to verify the presence of stored information. It is based on "testing", the ability to assign a variety of different values to bits which have not yet been input to the circuit by a specific time. The advantage in testing is that it provides a simple alternative to the sometimes arduous task of constructing fooling sets, as formalized in [U]. Unfortunately, these simple techniques seem to be inapplicable when inputs are read more than once. Therefore proofs in this paper are

based on counting arguments similar to fooling set constructions.

This choice of proof style has advantages. The techniques are more general, and, we believe, more insightful. For example, Theorem 2 not only provides the foundation for all our minimum area results, but more importantly, it establishes the central proof (and proof technique) for our AT^2 bounds.

In our VLSI model, the area measure is sufficient to store one bit per unit. I/O ports are counted as storage devices, and also require a unit of area. Wires have unit width and take unit time to transmit a bit, independent of length. The I/O schedule is assumed to be when- and where-determinate, i.e., the time and location for each input and output bit are prespecified. Our area bounds are proved for multilective input; that is, data can be read more than once. All computations and temporary storage of data must be done within the sorting device. We note that although our model is typical of VLSI characterizations, (and our language frequently refers to such a model,) the storage bounds are valid for a bit model sorting network, and, in fact, any digital sequential sorter.

For an essentially planar network model, such as VLSI, minimum area and minimum storage for sorting turn out to be within a constant factor of one another; for this reason, our minimum storage bounds are always denoted by A. In the AT^2 complexity measure, however, A denotes circuit area (not storage) and T denotes the circuit's running time; in this case, area can exceed storage by more than a constant factor. Also, the input schedule is required to be semelective in this case, that is, data is read only once. More complete information about these models can be found in [U].

The input variables are named $X_1, X_2, \ldots, X_n$. The sorted output variables are $Y_1, \ldots, Y_n$, where $Y_1 \leq Y_2 \leq \cdots \leq Y_n$. The bits of Y_j are $Y_{j,l}$, where $Y_{j,1}$ is the most significant bit (msb). The msb is sometimes called the 1'st most significant bit, and $Y_{j,k}$ is called a 1'st least significant bit (lsb). When $k > \log n$, we call the $\log n$ more significant bits of an X or Y variable the *more significant bits*, and define the *less significant* bits to have significance $1+\log n$ or more. Sometimes the distinction between more and less significant bits is made relative to $\log \frac{n}{r}$, rather than $\log n$.

Our basic proof strategy is to measure how much additional information is needed, in specific cases where a subset of the Y-bits are to be output, and a subset of the X-bits are known "for free."

A few definitions will help simplify the subsequent exposition. Let a circuit compute a boolean vector function $f: X \rightarrow Y$. Then $I(f(X))$ is the logarithm of the size of f's range:

$I(f)=\log|\{y|y=f(x)\}|$. If P is a predicate, then $I(f|P)$ is the logarithm of the size of f's range when the domain of f is restricted to $\{X|P(X)\}$. If $P=\{p_z\}_{z\in\zeta}$ is a partition of X, then $I(f|p_z)=I(f|X\in p_z)$.

Given n k-bit variables, $X=\{X_i\}_{i=1}^{n}$ and an n-tuple $\Pi=(x_1,x_2,\ldots,x_n)$ of κ-bit numbers, where $\kappa\le k$, we say Π is a *higher* (*lower*) *order assignment to* X if, for $i=1,2,\ldots,n$, the number comprising the κ more (less) significant bits of X_i equals x_i. Given a multiset Σ of n κ-bit numbers, where $\kappa\le k$, we say X has a *higher* (*lower*) *order restriction to* Σ, if the higher (lower) order assignment to X is an n-tuple comprising a permutation of Σ. If S is a set of n-tuples, X is restricted to S means the X's can be assigned any tuple in S. Given a subset χ of the bit variables in $\{X_i\}_{i=1}^{n}$, an assignment to χ is an assignment of values to χ. If R is an assignment to χ, and Σ is a multiset of n l-bit numbers, then $\Sigma|_R$ is the collection of X-assignments (n-tuples) comprised from permutations of Σ that agree with R on χ. Finally, the symbol $\circ$ is used to denote a kind of tuple concatenation: if $S=\{(a,b,c),(d,e,f)\}$, and $\Pi=(x,y,z)$, then $S\circ\Pi=\{(ax,by,cz),(dx,ey,fz)\}$, where ax means bit string a concatenated with bit string x.

2. Lower Bounds for Area

To simplify the exposition, we shall assume, in this section, that the I/O schedule inputs and outputs at most one bit per time step. Since I/O pads consume unit area, concurrent inputs can be viewed as serial inputs into a shift register. Our area (only) bounds never include area consumed by wires, and consequently there is no cost attributed to routing the stored inputs to their destinations. (Furthermore, such a routing would only affect the area by a constant factor.) Thus the area bounds for the simplified schedules also apply to circuits using arbitrary determinate schedules. Similarly, we may assume that input and output events never occur at the same time step. It should be noted that the input schedule is not assumed to be, say, r copies of a schedule where each input occurs once. Each bit of an input variable can have its r input instances read in any determinate way, and could, of course, have fewer than r instances.

Theorem 1.

A when- and where-determinate sorter of n k-bit numbers that inputs its data r times, where $2^k\le\frac{n}{r}$ satisfies $A=\Omega(2^k\log\frac{n}{r2^{k-1}})$.

Proof: It suffices to assume that $2^k\le n/2r$. Choose a time interval $\tau=[\tau_1,\tau_2]$ so that during the time $t\in\tau$, $n/2r$ lsb's are output by the circuit, and no more than $n/2$ least significant bits are input. Without loss of generality, we may assume that $n/4r$ of the bits output during τ

belong to variables $Y_{i,k}$, where $i \le n/2$. (An analogous argument holds for $i > n/2$.) Let those $n/4r$ lsb's comprise the set of variables $\Psi = \{Y_{i_1,k}, Y_{i_2,k}, \ldots, Y_{i_{n/4r},k}\}$. Name the inputs so that the least significant bits read during τ are $X_{1,k}, X_{2,k}, \ldots, X_{\alpha,k}$, where $\alpha \le n/2$. By assigning $X_i = 2^k - 1$ for $1 \le i \le \alpha$, we can guarantee that the outputs in Ψ are independent of the lsb's read during τ. The circuit must therefore have a memory at least as large as the information content of the outputs of these $n/4r$ least significant Y-bits. Formally, we assign values to the remaining $\{X_j\}_{j=\alpha+1}^{n}$ in batches. Put $i_0 = 0$, and let $i_l - i_{l-1}$ of these X's equal ξ_l, for $l = 1,2,\ldots,n/4r$. Set the remaining $n - \alpha - i_{n/4}$ X's to $2^k - 1$. Thus $Y_{i_l} = \xi_l$. Now the chip need not "remember" $n/4r$ distinct assignments to the lsb input variables $\xi_{l,k}$, because many of the ξ variables must have identical values (if $2^k \ll n/2r$). We complete the proof by assigning values to the ξ's in batches. Put $\eta = \frac{n}{4r2^{k-1}}$, and, for $j = 0,1,\ldots,2^{k-1}-1$, let $\xi_{j\eta}, \xi_{j\eta+1}, \cdots, \xi_{j\eta+\eta-1}$ have the number j comprising its $k-1$ most significant bits. This j'th batch has 0 assigned as the least significant bit for the first l_j ξ's, and 1 assigned for the $\eta - l_j$ remaining ξ's. Thus for each j, l_j is the number of output variables in Ψ, which have j assigned to the $k-1$ more significant bits, and which have 0 assigned to the least significant bit. Since l_j can be any value from 0 to $\frac{n}{r2^{k+1}}$, the number of such distinct outputs is at least

$$\left\lfloor 1 + \frac{n}{r2^{k+1}} \right\rfloor^{2^{k-1}},$$

whence taking the logarithm gives, $\Omega(2^k \log \frac{n}{r2^{k-1}})$. □

Theorem 2.

A when- and where-determinate sorter of n k-bit numbers that inputs its data r times, where $\frac{n}{r} < 2^k < (\frac{n}{r})^2$, satisfies $A = \Omega(\frac{n}{r} \log \frac{r2^{k+1}}{n})$.

Proof: Write $\eta = n/r$, $\kappa = \log\eta$, and $k = \kappa + \rho$. We now refer to the κ most significant bits of each input number as the *more significant*; the ρ least significant bits are *less significant*. In terms of the new variables, the area bound can be written as $A = \Omega(\eta\rho)$. It evidently suffices to have $\rho < \epsilon\kappa$, where ϵ is positive and will be specified later. In view of Theorem 1, we may also assume that $\rho > 6$.

We want to find a time interval during which a set Ψ, that comprises a large number of less significant bit variables, is output, but during which at most half of the more significant bits are read. An area bound can then be found by computing how much additional information must be remembered by the circuit, if its outputs are to be correct. Since there will be no

restrictions on which less significant bits are read during the interval, we bound the uncertainty in the output, essentially, by the somewhat informal expression:

$I(\Psi \mid$ all X-inputs except for $\frac{n\kappa}{2}$ more significant X-bits have fixed known assignments).

The principal difficulties in estimating this expression are that the known information can be asymptotically much larger than $|\Psi|$, and that some "large" sets of less significant bit variables are so highly correlated that their information content is too small to give an adequate estimate. Consequently, simple counting estimates seem to be insufficient for establishing the result, and a formal fooling set construction seems to be needed. Furthermore, some notion of the correlation among bit variables must presumably be included in the argument.

The construction has five steps:

(1) Find set Ψ of $\eta\frac{\rho}{6}$ less significant output bits that have a total correlation *cost* $<1/4$, and that are output in a time interval τ when at most $\frac{n\kappa}{2}$ more significant bits are input. (The correlation cost measure for an s-th significance output bit turns out to be 2^{-s}.)

(2) Assign a skewed distribution of values to the Y outputs so that the ordering of the Y's is independent of the assignments to the bits in Ψ. (Thus if the most significant bit variable that belongs to both Y_i and Ψ has significance s, then make sure (essentially) that $Y_{i+1}>Y_i+2^{k+1-s}$. The s more significant bits of Y_i are set accordingly, for $i=1,2,\ldots,n$.)

(3) Choose $n/4$ input variables that have the fewest number of more significant bits read during τ.

(4) Partition the selected input variables into sets having almost identical bit sets read during τ, and assign to large subpartitions output values that appear identical on the bit sets; then make the various lower significance bit assignments as uniformly distributed as possible.

(5) Show that because of the more significant X-bits which are not input during τ and the less significant bit assignments to the X's, the uncertainty of the less significant bit values for the Y's within each subpartition is high.

Let τ_h, $h=0,1,\ldots,6r$ be a partition of time so that during each of the time intervals $t\in(\tau_h,\tau_{h+1}]$, $\frac{\eta\rho}{6}$ bits, of significance $\kappa+1$ to k, are output. Let $C_h=\sum_{\substack{Y_{ij}\text{ output during}\\(\tau_h,\tau_{h+1}],\ s>\kappa}} 2^{-s}$. C_h is, essentially, the cost of having those lesser significant bits be independent.

Step 1. Clearly $\sum_{h=0}^{6r-1} C_h = \sum_{1\le j\le n} \sum_{\kappa<s\le k} 2^{-s} < \frac{n}{\eta} = r$, so at least $2r$ of the C_h's have values less than 1/4. Furthermore, during some time interval $\tau = (\tau_h, \tau_{h+1}]$ corresponding to one of those C_h's, no more than $\frac{n\kappa}{2}$ bits can be read, which are of 1 to κ significance, since the more significant bits are read a total of $rn\kappa$ times. Let that value of C_h be C. Let S_j be the set of significance positions of the less significant bits belonging to Y_j, which are output during τ, for $j=1,\ldots,n$, that is, the bit variables in $\{Y_{j,b}\}_{b\in S_j}$ comprise the less significant bits output during τ. This set comprises the less significant bit variables of step 1.

Step 2. We now assign values to the more significant bit positions of the Y outputs. Let $s_j = k+1-\min\{\infty, b\in S_j\}$. We temporarily set $Y_1=0$, and provisionally put, for $j=2,3,\ldots,n$

$$Y_j = \begin{cases} Y_{j-1}+2^{s_{j-1}}, & \text{if } s_{j-1} \ge s_j, \\ Y_{j-1}+2^{s_j} - \left(Y_{j-1} \bmod 2^{s_j}\right), & \text{if } s_{j-1} < s_j. \end{cases}$$

Evidently $Y_n \le 2^{k+2}C < 2^k$, so the Y's take on permissible values. Note also that $Y_j = Y_{j-1}$ only when $S_j = S_{j-1} = \phi$. Furthermore, the s_j least significant bits of Y_j can be altered arbitrarily without affecting the ordering of the Y's. Let the set V consist of all the Y_j's with $s_j>0$. Any $Y_j \in V$ currently has its s_j least significant bits set to 0. These lower significant bit assignments will be altered in Step 4. The assignments to bits of greater significance are permanent.

Step 3 begins with the observation that there are at least $\frac{n}{4}$ input variables, each having at least $\frac{\kappa}{3}$ more significant bits which are not input during τ. Now the number of different ways $\frac{\kappa}{3}$ bits can be distributed among the κ more significant positions is only $\binom{\kappa}{\kappa/3} < (\eta)^{\log 3 - 2/3} < (\eta)^{.92}$. If the X's are partitioned into "pools", where pool members have an identical set of $\frac{\kappa}{3}$ unread more significant bit positions, then the mean pool size is at least $\frac{1}{4}n^{.08}r^{.92}$. (If a variable has more than $\frac{\kappa}{3}$ more significant bits that are unread during τ, we can arbitrarily ignore all but a subset of size $\frac{\kappa}{3}$.) If we consider only X's belonging to pools of size $\frac{1}{8}n^{.08}r^{.92}$ or more, then at least $\frac{n}{8}$ X's remain.

Step 4. Values can now be assigned to the X's. We avoid matching difficulties by supposing that there are in fact more than n "good" ones, rather than just $\frac{n}{8}$. Accordingly, each

pool is replicated, say, 16 times. Later amends will trivially rectify this overcount.

Our strategy is to assign values to almost all the X's in a pool before going on to use the members in another pool. Accordingly, we pick a pool of X's, select a $Y_j \in V$, and put $s = s_j$. Let (the 0's in the binary number mask) B denote a set of $\frac{\kappa}{3}$ more significant bit positions, which are not input during τ, for members of the X-pool. Let μ be the number of $Y_k \in V$, which have the same bit values as Y_j on the unmasked bits in B, and for which $s_k = s_j$: $\mu = |\{Y_k \in V \mid Y_k \wedge B = Y_j \wedge B,\ s_k = s\}|$. If $\mu < 2^s$, the process is repeated with another Y selection. Otherwise, the matching Y's are **removed** from V in groups of 2^s, and corresponding X's are removed from the pool. Each X is assigned the $k-s$ leading bits of its corresponding Y. The less significant bits of a 2^s-member X-group are assigned some permutation of the respective s-bit values $0,1,\ldots,2^s-1$. If the number of matching Y's is not a multiple of 2^s, it suffices to leave that last remnant Y group in V.

When an X pool has fewer than 2^s remaining members, it is discarded and a new pool (of size $\geq 2^\rho$) is started along with new selections of Y's from V. Since we have assumed that the number of X pools is sufficient, the process will only terminate when no $Y \in V$, for the selected (mask defining the) X pool, gives a sufficient count for μ. In that case, the total number of Y's remaining in V is small. For each value s, the X pool (or more precisely the mask) partitions the set $\{Y_k \in V : s_k = s\}$ into $2^{k-s-\kappa/3}$ equivalence classes. Evidently each class contains fewer than 2^s Y's. Thus the number of less significant output bits, which are output during τ, and which belong to the Y's left in V, is bounded by

$$\sum_{s=1}^{\rho} 2^s 2^{k-s-\kappa/3} s < \frac{2^k \rho^2}{\eta^{1/3}} = \rho^2 2^\rho \eta^{2/3}.$$

If we require $\rho 2^\rho < \frac{1}{12}\eta^{1/3}$, then at least $\eta\frac{\rho}{12}$ output bits will be assigned values by this process. The other constraint on ρ is that the pool size be large enough to have 2^s members: $2^s \leq 2^\rho \leq \frac{1}{8} n^{.08} r^{.92}$. Thus, it is sufficient to set $\epsilon = .07$, so that $\frac{n}{r} < 2^k < (\frac{n}{r})^{1.07}$ (and have $r \geq 8$ or n large).

The accounting for the excess pools of X's is now simple. For each of the 16 multiples of a pool, keep the one whose corresponding Y set has the maximum number of less significant bits that are output during τ. Let the set V comprise those Y collections. Thus, the Y's in V have at least $\eta\rho/192$ less significant output bits which can have various values, depending on the unread X bits, and which are output during τ. All remaining Y's are paired with unmatched (genuine) X's, which are assigned the value of their corresponding Y's.

Step 5. By construction, $V=\cup Z_\lambda$, where the union is over disjoint sets, and each Z contains (for some s) exactly 2^s of the Y_u's, each of which has $s_u=s$. Moreover, those Y's can have their s least significant bits assigned any permutation of the s-bit numbers $0,1,\ldots,2^s-1$, depending on unread bit values of the X's. Let b_λ be the number of less significant bits that belong to the Y's in Z_λ, and that are output during τ: $b_\lambda = \sum_{Y_j\in Z_\lambda} |S_j|$. Let n_λ be the number of different possible output configurations (assignments) for the b_λ bits. Then

$$n_\lambda \geq \frac{2^s!}{2^{s2^s-b_\lambda}} > \frac{2^{s2^s}e^{-2^s}}{2^{s2^s-b_\lambda}} = e^{-2^s}2^{b_\lambda} > 2^{b_\lambda - 1.5\times 2^s}.$$

Therefore, if each $Y\in Z_\lambda$ has, on the average, at least 2 output bits, so that $b_\lambda \geq 2^{s+1}$, then $b_\lambda \geq .25b_\lambda + 1.5\times 2^s$, whence $n_\lambda > 2^{.25b_\lambda}$. If $b_\lambda < 2^{s+1}$, then an adequate estimate for n_λ is based on the fact that exactly 2^{s-1} of the s'th least significant bits will be 1's:

$$n_\lambda \geq \binom{2^s}{2^{s-1}} \geq 2^{2^{s-1}} > 2^{.25b_\lambda}, \qquad b_\lambda < 2^{s+1}.$$

It follows that due to the uncertainty of the more significant X-bit values that are not read during τ, the total number of different assignments to the less significant bits that are output during τ is at least $\prod_\lambda n_\lambda \geq 2^{\frac{np}{768}}$, and hence $A = \Omega(\frac{n}{r}\log\frac{r2^{k+1}}{n})$. □

It is worth summarizing what we have proved. Let W_κ be the set of masks that correspond to the $n\times k$ bits of the input variables, and that have $\frac{n\kappa}{2}$ 1's among the $n\kappa$ more significant bits, and all 1's among the $(nk-n\kappa)$ less significant bits (i.e., all less significant bits are revealed). Let $Q_{\kappa,r}$ be the set of all partitions of these less significant output bits that comprise r subsets. Given a partition $q\in Q$ (with subscripts κ and r understood), we represent a set $q_l\in q$ by its mask. We have proved, in Theorem 4, that for a sorting function f, (where $n/r<2^k<(n/r)^2$,)

$$\exists\kappa\ \forall q\in Q\ \exists q_l\in q\ \forall w\in W\ \exists z: I\left(f(X)\wedge q_l | X\wedge w=z\right) = \Omega(\frac{n}{r}\log\frac{r2^{k+1}}{n}).$$

Furthermore, this bound for I is achieved when $\kappa=\log\frac{n}{r}$. A more precise statement would include the fact that the assignments for the more significant X-bits can restricted to permutations of a multiset that is existentially quantified at the innermost level, and that is known (i.e. has its values given as part of the conditional expression for I).

It should be noted that the proof is somewhat simpler when r is set to 1, since each X can then receive a unique value for its $\log n$ more significant bits. In this case, the less significant bits are completely uncorrelated; they can have arbitrary assignments without changing the permutation. The consequence of all this is that the variable s in the proof of Theorem 2 can be fixed at $s = k+1-\log n$. Let $k \in [1+\log n, 1.07\log n]$, and let W be the set of masks that correspond to the $n \times k$ input bits, and that have $\frac{n\log n}{2}$ 1's among the $n\log n$ more significant bits, and all 1's among the less significant bits. Let Q be the set of all subsets containing half of the less significant output variables. We again represent $q \in Q$ as a mask, and take $|q|$ to be the number of ones in the mask (i.e., the number of bits revealed), so that $|q| = n(k-\log n)/2$. From Theorem 1, we have that for a sorting function f,

$$\forall q \in Q, w \in W \; \exists z: I\left(f(X) \wedge q \,|\, X \wedge w = z\right) = \Omega(|q|).$$

It is reasonable to ask if the above bound would hold if Q were to contain contain all subsets of less significant output bits. The answer turns out to be affirmative, although we will show, in Corollary 1, a somewhat weaker result, which is adequate for our purposes. In [S2], the following stronger result is established:
For $k \in [1+\log n, 2\log n]$, and Q the power set of the less significant output bits,

$$\forall q \in Q, w \in W \; \exists z: I\left(f(X) \wedge q \,|\, X \wedge w = z\right) \geq |q|(.5 - o(1)).$$

Furthermore, the constant .5 is the best possible (although the function $.5|q|$ is only tight up to a factor of 2 for arbitrary $|q|$.).

The weaker result will follow from our construction for the case $k = 1.07\log n$ (and $r = 1$). Given a set χ of $\frac{n\log n}{2}$ more significant input bits, the construction, in this can be used to find an assignment R on χ, the multiset $U = \{0,1,2,\ldots,n-1\}$, a lower order assignment L, and a rectangular set of less significant output bit variables $\Psi = \{Y_{j,l} \,|\, j \in J, \; l \in [1+\log n, k]$, where $J \subset [1,n]$ and $|J| = n/32$, such that

$$I(\; \Psi \;|\; (X_1, \ldots, X_n) \in U|_R \circ L \;) \geq .25|\Psi|.$$

Moreover, the uncertainty in the values of Ψ will be a uniform .25 per bit: if $\zeta \subset \Psi$, then

$$I(\; \zeta \;|\; (X_1, \ldots, X_n) \in U|_R \circ L \;) \geq .25|\zeta|.$$

We deduce

Corollary 1.

Suppose $n < 2^k \leq n^2$. Let χ be a set of $\frac{n\log n}{2}$ input bit variables, each belonging to the $\log n$ more significant positions, and let Ψ be a set of output bit variables, which is contained

in $\{Y_{i,l} \mid n/3 \le i < 2n/3,\ \log n < l\}$. Then there is a multiset, U, of n ($\log n$)-bit numbers, a lower order assignment, L, of n ($k-\log n$)-bit numbers to X, and an assignment, R, of values to χ so that

$$I(\ \Psi \mid (X_1, \ldots, X_n) \in U|_R \circ L\ ) = \Omega(|\Psi|).$$

Proof: Note that the output bits Ψ belong to numbers ranking in the middle third of the n values. Furthermore, they are of sufficiently low significance that they need not affect the ordering of the Y's.

Put $\nu = n/3$, and pick $\lambda \in [0, 2\nu]$. We consider the problem of sorting $X_1, X_2, \ldots, X_\nu$ into $Y_{1+\lambda}, Y_{2+\lambda}, \ldots, Y_{\nu+\lambda}$. Suppose, for the moment, that $k = 1.07 \log \nu$. A fooling set construction is as follows. We rename the X's, if necessary, so that $X_1, X_2, \ldots, X_\nu$ have at most $\frac{n \log n}{6}$ bits in χ. We set, for $j > \nu$, $2\nu - \lambda$ of the X_j's to $2^k - 1$, and λ of them to 0. As described in the remarks following Theorem 2, we construct a Cartesian rectangle $\overline{\Psi}$ of $\frac{.07 \nu \log \nu}{32}$ less significant output bits, a multiset $U = \{0, 1, 2, \ldots, \nu - 1\}$, an assignment R on χ, and a lower order assignment L, such that

$$I(\ \overline{\Psi} \mid (X_1, \ldots, X_\nu) \in U|_R \circ L\ ) \ge .25|\overline{\Psi}|.$$

The point of this construction is that the choice of lambda merely shifts $\overline{\Psi}$ with respect to the Y's. Specifically, for $\lambda = 0$, $\overline{\Psi} = \{Y_{j,1+\log \nu}, Y_{j,2+\log \nu}, \ldots, Y_{j,1.07 \log \nu}\}_{j \in \Lambda}$, for some set $\Lambda \subset \{1, 2, \ldots, \nu\}$, $|\Lambda| = \frac{\nu}{32}$. For general λ, $\overline{\Psi}(\lambda) = \{Y_{j,1+\log \nu}, Y_{j,2+\log \nu}, \ldots, Y_{j,1+\log \nu}\}_{(j-\lambda) \in \Lambda}$. By averaging over λ, it is easy to show that for some shift λ_0, $|\overline{\Psi}(\lambda_0) \cap \Psi| \ge |\Psi|/64$. The set $\overline{\Psi}$ in this construction is easily modified for the general case $n < 2^k < n^2$, with the result that $|\overline{\Psi}(\lambda_0) \cap \Psi| = \Omega(|\Psi|)$, whence the result follows. □

Curiously, Corollary 1 appears to be too weak to prove a good area bound in the case $(n/r)^2 \le 2^k \le n$, which is the subject of Theorem 3. The obvious application is to combine it with the fact that there must be a suitable interval of time during which at most $\frac{n \log n}{2}$ more significant bits are input, and during which $n\frac{k - \log n}{6r}$ suitable less significant bits, Ψ, are output. The conclusion would be that $A = \Omega(\frac{n}{r} \log \frac{2^{k+1}}{n})$, which is a bit weak (the best bound is $\Omega(\frac{n}{r} \log \frac{r 2^{k+1}}{n})$), and does not hold for the desired range of k.

Theorem 3.

A when- and where-determinate sorter of n k-bit numbers that inputs its data $r \le n$ times, where $(\frac{n}{r})^2 \le 2^k \le n$, satisfies Then $A = \Omega(\frac{kn}{r})$.

Proof: Put $\kappa=\frac{2k}{3}$, $\eta=2^{\kappa}$. We will apply Corollary 1 to a sorting problem having η Y's and X's.

Standard arguments show that there is an interval of time τ, when no more than $\frac{kn}{3}$ bits are read, which belong to the κ more significant positions, and when $\frac{kn}{18r}$ bits are output, which belong to the outputs $Y_{i,j}$, where $n/3\le i<2n/3$, and $l>\kappa$. Let the Y's having these less significant output bits comprise $\{Y_\lambda\}_{\lambda\in\Lambda}$. Clearly $|\Lambda|\le\frac{kn}{18r}\le\frac{k2^{\frac{k}{2}}}{18}$. Furthermore, it is easy to check that $\frac{k2^{\frac{k}{2}}}{18}<\frac{2^{\kappa}}{3}$. So $|\Lambda|<\frac{\eta}{3}$. Let $\Lambda_1=\{1,2,\ldots,\lfloor(\eta-|\Lambda|)/2\rfloor\}$, and $\Lambda_2=\{\frac{2n}{3},\frac{2n}{3}+1,\ldots,\frac{2n}{3}+\lceil(\eta-|\Lambda|)/2\rceil-1\}$. Put $\bar{\Lambda}=\Lambda_1\cup\Lambda\cup\Lambda_2$. It suffices to assume that $n>3$, whence $\frac{n}{3}>\frac{\eta}{2}>\frac{\eta-|\Lambda|}{2}$. This inequality ensures that the indices in Λ are between the indices of Λ_1 and Λ_2. Clearly $|\bar{\Lambda}|=\eta$. We rename the X's, if necessary, so that $X_1,X_2,\ldots,X_\eta$ have at most $\frac{\eta\kappa}{2}$ bits which are input during τ, and which belong to the κ more significant positions. This is possible since on the average, each X has only $\frac{\kappa}{2}$ more significant X-bits read during τ. We apply Corollary 1 to this reduced problem, and conclude that $A=\Omega(\frac{kn}{r})$ for sorting $\{X_l\}_{l=1}^{\eta}$ into $\{Y_\lambda\}_{\lambda\in\bar{\Lambda}}$.

To complete the proof, we must ensure that the larger problem contains our specific η-number sorting problem. This is easy. It suffices to give the remaining inputs, $\{X_l\}_{l=\eta+1}^{n}$ fixed values, assigned in batches. Let $\bar{\Lambda}=\{\lambda_l\}_{l=1}^{\eta}$, where $\lambda_1<\lambda_2<\cdots<\lambda_\eta$. Put $\lambda_0=0$. Let $n-\lambda_\eta$ X's be 2^k-1, and, for $i=1,2,\ldots,\eta$, set $\lambda_l-\lambda_{l-1}-1$ X's equal to ξ_l. Define the ξ's so that $\xi_{l,j}=0$ for $l>\kappa$, and $\xi_{l,j}=Y_{\lambda_l,l}$ for $l\le\kappa$. It is easy to see that these X's have the right rankings; they fill the Y gaps correctly. □

It should be observed that in Theorem 3, we construct a permutation on a thin set ($|\Psi|\ll n$). The construction given in Theorem 2 cannot be directly applied in this case (unless, for technical reasons, it turns out that $|\Psi|$ is extremely small) because almost all of the output bits may be mostly "wasted." Theorem 2 succeeded by constructing a permutation only among Y's having output bits in Ψ; no Y's without such timely less significant output bits were used in the permutation construction. The difficulty with using these "empty Y's" is to make certain that they are not overcommitted, that is, to ensure their less significant assignments are not needed by members of Ψ assigned to many different X "pools".

Theorem 2 and Theorem 3 can be combined to read,

Corollary 2.

A when- and where-determinate sorter of n k-bit numbers that inputs its data $r \leq n$ times, where $\frac{n}{r} < 2^k < n^{O(1)}$, satisfies $A = \Omega(\frac{n}{r} \log \frac{r2^{k+1}}{n})$. □

The following lemma improves our lower bound when r is large.

Lemma 1.

For any number of replicated inputs, the area of a circuit that sorts n k-bit numbers must satisfy, $A = \Omega(\log n)$.

Proof: The reason for this bound is that a circuit that inputs rnk bits and outputs nk must have enough internal states to be able to count, effectively, to $\min(r,n)$. Adding $\log r$, for $r \leq n$, to the bounds in Theorem 1 or Corollary 2 is equivalent to adding $\Omega(\log n)$. □

3. AT^2 Bounds

The proof of Theorem 2, in the case $r = 1$, shows

Theorem 4.

Suppose $n \leq 2^k \leq n^{O(1)}$. Then a when- and where-determinate circuit that sorts n k-bit numbers and reads the inputs once must satisfy $AT^2 = \Omega(n^2 \log^2 \frac{2^{k+1}}{n})$.

Proof: We may assume $k \geq 1 + \log n$, as otherwise we can replace n with $n/2$. We can also assume $T < \frac{n}{3} \log \frac{2^k}{n}$, as otherwise the result is trivially true. The chip is partitioned in the standard manner so that each side of the circuit outputs at least $\frac{n}{3} \log \frac{2^k}{n}$ bits, which are of significance $1 + \log n$ or below. At least half of the more significant bits (*i.e.*, of $\log n$ significance or better) must be input on one side of the circuit, say, the left. Then the proof of Theorem 2 shows that the right side needs $\Omega(n \log \frac{2^k}{n})$ bits of information from the left, if the circuit is to sort correctly. □

Note that the input distribution of less significant X-bits has not been specified, insofar as the partitioning line (as well as the choice of sides) is concerned. The proof thus permits the right side of the circuit to know all of these assignments! Since we are using an information theoretic (i.e. counting) argument, we may suppose that the left side is an oracle, and knows the values of all the input bits. The information flow is now unidirectional; we are estimating the amount of essential information that the right side of the circuit is still missing. The point of this formulation is that these "benefits of the doubt" are of no help whatsoever (up

to a constant factor). The information that must cross the partition is proportional to the information content of the less significant output bits. Moreover, we need no new proof to establish the fact.

For completeness, we note that in [S1], a multiple cut argument is used to prove

Theorem 5 [S1].

Suppose $2^k \leq n$. Then a when- and where-determinate circuit that sorts n k-bit numbers and reads the inputs once must satisfy $AT^2 = \Omega(n2^k)$. □

When $k \leq (1+o(1))\log n$, there are actually two AT^2 bounds. We have as yet made no restrictions on how the input and output ports are distributed in the sorting circuit. For VLSI models, the I/O ports are typically required to be located along the perimeter of the circuit (or more precisely, the convex hull). We may assume, without loss of generality, that the circuit is rectangular, with length L, and width $W \leq L$. Ullman [U] points out that in cases restricted to perimeter I/O, there is an LT bound based on the requirement to read the the input data: $4LT = \Omega(nk)$. If the information that must flow across the circuit is I, then taking a Thompson cut along the shorter direction, gives $WT = \Omega(I)$, so that $AT^2 = \Omega(Ink)$. Our contribution, in this case, is the determination of I. The arguments for Theorem 4 gives, $I = \Omega(n\log\frac{2^{k+1}}{n})$. Consequently, we have

Theorem 6.

Suppose $n \leq 2^k \leq n^{O(1)}$. Then a when- and where-determinate circuit that sorts n k-bit numbers, reads its inputs once, and has its I/O ports along its perimeter must satisfy $AT^2 = \Omega(kn^2\log\frac{2^{k+1}}{n})$. □

In particular, as shown in [U], for $k = \log n$, $AT^2 = \Omega(n^2\log n)$.

Additional results on AT^2 lower bounds for sorting can be found in [S2], and optimal sorters for various length numbers are constructed in [CS].

4. Upper Bounds

Circuits for sorting n k-bit numbers were presented in [S1]. Theorems 7 and 8 are simple applications of the sorting schemes given there. The only difference is that since the inputs can be read r times, it suffices to use the sorting schemes to output, for input phase j, $Y_{1+jn/r}, Y_{2+jn/r}, \ldots, Y_{n/r+jn/r}$, for $j = 0,1,\ldots,r-1$. Such a circuit requires sufficient size to contain pass j's intended output of n/r sorted k-bit numbers, the (largest) X-index of the maximum value so stored, a counter possibly going up to n, a small amount of controller logic, and, say, two temporary registers of k- and $\log n$-bits. As a consequence of the data storage

schemes in [S1], it follows that

Theorem 7.

For input replicated r times, a set of n k-bit numbers, where $2^k \leq \frac{n}{r}$, can be sorted in area $A = O(2^k \log \frac{n}{r2^{k-1}} + \log n)$.

This result is clearly optimal for all values of r.

Theorem 8.

For input replicated $r \leq n$ times, a set of n k-bit numbers, $\frac{n}{r} \leq 2^k \leq n^2$, can be sorted in area $A = O(\frac{n}{r} \log \frac{r2^{k+1}}{n} + \log n)$.

This result is also optimal.

The next sorting construction is optimal for some values of r and k, and provably suboptimal for others.

Theorem 9.

For input replicated $r \leq n$ times, a set of n k-bit numbers, where $2^k > n^2$, can be sorted in area

$$A = \Theta(\frac{n}{r} \log n) + O(G),$$

$$\text{where } G = \min \{n, \frac{kn}{r}, \frac{n}{r^{.5}} \log n\}.$$

Discussion: For $r < \log n$, n is dominated by $\frac{n}{r} \log n$, and hence the area bound is tight for $r = O(\log n)$. The first term in the expression for G comes from using n input pads, one for each number, and marking which inputs have been already sorted. If other pad markings indicate which inputs are candidate maximum values for the current $\frac{n}{r}$ group of values currently being sorted, and which are evidently smaller values in the group, then only $O(n)$ bits are needed in addition to the number necessary for a rank and forward scheme on $\frac{n}{r}$ numbers. The second term arises from the obvious approach of storing the next $\frac{n}{r}$ numbers to sort. The third results from intermediate passes to locate the addresses (indices) of the current $\frac{n}{r^{.5}}$ numbers to sort. In this case, all pairs of numbers are compared in each of $r^{.5}$ sorting phases, and the addresses of suitably ranked values are stored.

A natural question arises as to the why the gap occurs for sorting with repeated inputs and large k. Part of the discrepancy is caused by the format of the output schedule, and the failure of our techniques to capture and to quantify adequately two constraints inherent to the format. If output were not required to be when determinate, but instead were required to comprise ordered pairs (i,x), where, i is the rank of input value x, then the solution would be meaningful, and the circuit would certainly be a sorter. It is not difficult to construct a circuit which sorts in this manner, reads the input r times, and which uses an area proportional to $\frac{n}{r}\log n$ area. Furthermore, if such sorted data were then used as input, another kind of sorter could be designed that reads its data r times, produces the standard where- and when-determinate output, and that has area $A=\frac{n}{r}\log n$. Both tasks are easy. The obvious composition, however, takes r^2 input passes for area $\frac{n}{r}\log n$. The result is an alternative construction giving the $\frac{n}{r^{.5}}\log n$ term for G in Theorem 9.

Also, the information theoretic nature of the proofs ensures that for replicated inputs, the lower bounds arguments would hold if an oracle were consulted r times. A good oracle, for example, might return the addresses of the next $\frac{n}{r}$ numbers to output, whence the requisite area would be $O(\frac{n}{r}\log n)$. This bound accounts for an $O(\frac{n}{r}\log\frac{n}{r})$ rank and forwarding cost plus an $O(\frac{n}{r}\log r)$ area to store the addresses. Evidently, a more delicate decomposition of time and information flow is needed to establish better lower bounds in this case.

We now describe a sorting scheme that reduces the bounds gap for some cases.

4.1. Partitioning techniques for multilective sorting

In [S1], a rank and forward sorting scheme is shown to work well when the number of bits is large, say, for $k>2\log n$. An essential aspect of the scheme, however, is to be able to input all the numbers in bitwise parallel. The purpose of this section is to address the problem of rank and forward schemes when the area is less than n. As in the previous section, the sorting will proceed on consecutively ranking values. The problem is how to determine which inputs to sort in a given pass, so that rank and forward sorting can be applied to the selected variables.

Our rank and forward sorting for large $r<n$ is based on partitioning. We suppose the scheme sorts x numbers per phase. Evidently, it suffices to know the addresses of the x numbers for the current phase. The problem is how to identify them, and how to remember

them. We describe methods which store the addresses of the numbers to be sorted in the current phase. In view of Theorem 9, these addresses can be stored without appreciably increasing the area required to sort; the real difficulty is how to identify the numbers. One way to identify them is by ranking all n numbers during each phase. The addresses are likely to require storage even for this naive sorting scheme if the output is to be when-determinate, since specific rankings might be discovered at an indeterminate time, when not all the bits are available or are scheduled to be output. This method can be implemented by comparing all pairs of numbers. One possible implementation is outlined in the remarks following Theorem 9.

We now describe sorting algorithms which are better, at least when k is not too large. They will, however, require $\Omega(k)$ storage. Since the methods are somewhat complicated, inferior algorithms will be presented first, and modifications will be subsequently outlined and analyzed.

In the following discussion, it is convenient to assume that all input values are distinct. This assumption causes no difficulty, since the sorting concerns the case when $k>2\log n$, and we can then use lexicographical sorting with the index (address) of each input variable as the secondary key.

Evidently the relevant addresses can be computed in one pass if the the largest and smallest values in the current sorting phase are known.

A reasonable scheme to sort x numbers per pass might be to initially store the addresses of the h'th largest numbers, for $h = x,2x,4x,8x,\ldots$. We call such a sequence an *ordered partitioning branch*. These values could be found via techniques to compute order statistics. Then only a small amount of computation might be necessary to update the sequence for each new sorting phase.

It turns out that this scheme can be implemented. Improvements are based on modifying the standard order statistic algorithms, relaxing the requirement that they find exact order statistics, and relaxing the requirement that the rankings of the elements (represented) in the ordered partitioning branch be exact.

Given a set S, define $S_{x\,y}$ to be $\{s \in S | x<s\leq y\}$. Given a finite set S with $|S|=c$, we call a sequence $-\infty\leq s_0<s_1<s_2<\cdots<s_d<s_{d+1}=\infty$, an *x-k-decomposition* of S if

(1) $s_0=-\infty$, if $k=0$, and s_0 is the kx'th smallest member of S, if $k>0$.

(2) $\{s_1,\ldots s_d\}\subset S$.

(3) $3x > |S_{s_0 s_1}| \geq x$.

(4) For $i=1,\ldots,d$, $c2^{-l} \geq |S_{s_{l-1} s_l}| > c2^{-l-1}$ implies $c2^{-l+1} \geq |S_{s_l s_{l+1}}| > c2^{-l-1}$.

(5) For $i=0,1,\ldots,d-2$, $c2^{-l} \geq |S_{s_l s_{l+1}}| > c2^{-l-1}$ implies $|S_{s_{l+3} s_{l+4}}| > c2^{-l}$.

The addresses of the values $s_0, \ldots, s_d$ will form our (generalized) x-k-ordered partitioning branch of S. The requirements 1 through 5 have simple interpretations. They are

(1) s_0 is the largest value sorted so far (or $-\infty$, if none have been sorted).

(2) The values are in S.

(3) The partition of S with the smallest values contains enough values for the current output phase, and not too many more.

(4) If we call the half open intervals $(c2^{-l-1}, c2^{-l}]$ slot l, then the sizes of two consecutive partitions in the ordered partitioning of S belong to the same or consecutive slots.

(5) The sizes of at most three (consecutive) partitions of S can belong to a single slot.

We remark that it will soon be convenient to relax the definition of an x-k-ordered partitioning to allow some empty slots. The reason for this next (and last) modification is to define an invariant that allows an ordered partitioning to be updated in each sorting phase with only one application of a subroutine call. The subroutine will, given addresses of partition values s_l and s_{l+1}, return an address of a value $\hat{s}$, where $s_l < \hat{s} < s_{l+1}$, and $\frac{2}{3}|S_{s_l s_{l+1}}| \geq |S_{\hat{s} s_{l+1}}| \geq \frac{1}{2}|S_{s_l s_{l+1}}|$. Thus we need a separator algorithm which partitions a set into two pieces, and the larger piece contains the larger values and has a size at most two thirds of the original set.

We postpone a discussion of the separator subroutine to concentrate on the other aspects of the algorithm. The separator is used, of course, to split a subset $S_{a\,c}$ into two pieces, $S_{a\,b}$ and $S_{b\,c}$. It is simple to verify that if $S_{a\,c}$ occupied slot $l-1$, then there are only three possibilities for the new pieces. Either $S_{b\,c}$ occupies $l-1$ and $S_{a\,b}$ occupies l, or both occupy slot l, or $S_{b\,c}$ occupies slot l, and $S_{a\,b}$ occupies $l+1$. The point of this organization will be that in any case, the formerly empty slot l will get a subset.

The sorting algorithm is:

Create x-0-ordered partition branch;
While not done do
    Sort x smallest numbers remaining in a partition;
    Call separation algorithm and fill lowest empty slot (if any) by

splitting the lowest partition in the next larger slot

End.

We identify as the *bottom slot* a pair of slots, namely the slot containing $S_{s_0 s_1}$ and the next larger slot. Higher slots remain uniquely named. We are willing to sort all the members of a single partition (having the smallest values) in the bottom slot, if the partition contains x or more members. Otherwise we merge the set with the next piece and sort that. Correctness follows from the following invariant. Between closest (internal) empty slots is a slot containing two or more subsets. That is, if slots j and $l>j$ are empty (and above the bottom slot), then either there are two subsets with sizes in slot h, for some $h: j<h<l$, or all members of the partition have sizes belonging slots below j. A simple induction argument will show that this property is preserved. It is clearly sufficient to guarantee that two (internal) empty slots cannot be consecutive, whence the lowest empty slot can always be filled. In particular, the bottom slot will never be empty (until all the numbers are sorted).

Creating the initial ordered branch is also simple. The details are omitted. We now consider separator algorithms.

Suppose we read l numbers at a time, and use sorting circuitry to find the median of the input. The standard algorithm [AHU] for $l=5$ reads,

OrderS(S, m)

If $|S|\leq l$ Return (m'th largest value in S);

Partition S into $|S|/l$ pieces of size l;

$\hat{S} \leftarrow \cup\{\text{median of each piece}\}$;

$\mu \leftarrow \text{OrderS}(\hat{S}, \frac{|\hat{S}|}{2})$;

$S_l \leftarrow \{s \in S | s < \mu\}$;

$S_e \leftarrow \{s \in S | s = \mu\}$;

$S_g \leftarrow \{s \in S | s > \mu\}$;

$m_l \leftarrow |S_l|$;

$m_e \leftarrow |S_e|$;

If $m_l \geq m$ Return($OrderS(S_l, m)$)

Else if $m_e \geq m - m_l$ Return($OrderS(S_e, m - m_l)$)

Else Return($OrderS(S_g, m - m_e - m_l)$);

End.

This method will be modified as follows:

(1) Approximate order statistics are found, rather than exact ones.

(2) A depth first iteration is used, instead of breadth first recursion.

(3) Rather than return a median from one level to use as a separator at the next, separators from the deepest level are returned to the top level.

The consequences are

(1) A savings of input passes

(2) A savings of input passes and area

(3) A savings of input passes at a cost of finding a very weak (preliminary) separator.

Then l can be tuned for the best performance.

First, though, it should be noted that the standard algorithm as given is materially simplified when l is chosen optimally. The reason is that if the algorithm finds medians of l numbers at a time, then $\frac{n}{l}$ median addresses must be stored, whence a good separator can be found in 2 passes, and the area for this portion of the algorithm is $l+\frac{n}{l}\log l$. If x numbers are sorted per phase, then the circuit can be implemented in area $A = O(k+x\log n+l+\frac{n}{l}\log l)$ and input passes $r = O(\frac{n}{x})$. We set $l=(n\log n)^{.5}$, whence $A = O(k+(n\log n)^{.5}+\frac{n}{r}\log n)$.

One difficulty with the algorithm as outlined is that it is not where- determinate. The difficulty occurs at the outer level. When only a subset of the n numbers are to be processed (due to partitioning), the $\frac{n}{l}$ medians found in the first pass will not be medians of uniformly sized subsets if the outer level is to be completed in one pass. Thus either additional passes are needed, or weighted median techniques must be used.

It turns out that weighted medians can indeed be sensibly used, and the area bound is affected by no more than a constant factor. The details are omitted.

In the modified depth first method, $\log_l n$ groups of l addresses are stored, one group for each level of depth in the algorithm. The following skeletal description shows the organization, but omits critical details essential to correctness and efficient operation.

```
Algorithm Weaksep
    Procedure Load(k,d)
    If d=0, k ← address of median of l current values;
        Load(k,1)
    Else insert k into addressbucket(d);
```

If addressbucket(d) is (possibly) full then

$k \leftarrow$ address of median of values represented in addressbucket(d);

Load(k,d+1);

Return;

End Load;

Repeat

While a group of l inputs are left unread Do

Load(k,0);

Flush lower levels (or assume $n = l^x$).

Return final address.

Partition based on value at address;

With bigger subpiece until suitable separator found;

End Weaksep.

It should be noted that due to when-determinism of the input, separator computations at level d require l^d values to be read, so that all values corresponding to the stored addresses can be compared. At the top level, the separator is only computed for a subset of the inputs, which lie between two partitioning values. Consequently, a when determinate operation of this scheme that uses input passes efficiently will empty buckets which are not full. As a consequence, the median computations are weighted. The partitioning values that are found are very weak separators. Indeed, after $\log_l n$ levels of recursion, a weak separator is found which partitions the original set into two pieces, neither smaller than $n^{1-\log_l 2}$. It is easy to see that if the set containing the median is partitioned further, and the process repeated, then a strong separator, which divides the set into two pieces, none smaller than one fourth of the original set, is found in $(\frac{n}{2})^{\log_l 2}$ iterations. Moreover, a 2/3 -- 1/2 separator can be constructed by applying a small number of calls to a separator algorithm that partitions sets into pieces of size 1/4 or larger.

If values (rather than addresses) were stored, Weaksep could be implemented with the input read only once. When addresses are stored, a weak separator can be found in $\log_l n$ passes over the input, which corresponds to a (distributed) input pass for each bucket level. Hence $O((\frac{n}{2})^{\log_l 2} \log_l n)$ input passes are needed per partitioning phase.

Recalling that l numbers valued 0 to $n-1$ can be stored in $l \log \frac{n}{l}$ area, and that the depth of the separator finding scheme is $\log_l n$, we conclude that the area for the separation

operation is $l\log\frac{n}{l}\log_l n$. If x numbers are to be sorted in each sorting phase, then their addresses can be found in one input pass if surrounding values are known. The area to store the addresses is $O(x\log\frac{n}{x})$. The sorting can be done in one more input pass. Our ordered partition stores $\log\frac{n}{x}$ partitioning addresses, and is updated only once per sorting phase. Thus the total number of input phases (including initialization passes) is $r=O(n^{\log_l 2}\log_l n\log\frac{n}{x}+\frac{n}{x}n^{\log_l 2}\log_l n)$ and the area is $A=O(k+x\log n+l\log n\log_l n)$. Set $l=2^{\frac{1}{\delta}}$, where δ is fixed. Then $A=O(k+\frac{n^{1+\delta}}{r}\log^2 n)$.

If $r=n^\sigma$ where $\sigma<1$, setting $l=\frac{n}{r}$ gives, $A = O(k+\frac{2^{\frac{1}{1-\sigma}}}{1-\sigma}\frac{n}{r}\log n)$. The corresponding expression is a bit tamer for $l=2n/r$.

Theorem 10.

For input replicated $r\le n^\sigma$ times where $\sigma<1$, a set of n k-bit numbers can be sorted in area $A=O(\frac{n}{r}\log n + k)$. □

Evidently, if $k=O(\frac{n}{r}\log n)$, then such a sorting circuit can be implemented with an area that is within a constant factor of optimal.

5. Conclusions

We have shown that for inputs read $r\le n$ times, n integers in the range $[0,m-1]$ can be sorted in area

$$A(n,m,r) = \Theta\begin{cases}\log n + m\log\frac{2n}{rm}, & m\le n/r,\\ \log n + \frac{n}{r}\log\frac{2rm}{n}, & n/r\le m<n^2.\end{cases}$$

These bounds also apply to a bit modeled sorting network, since the sorters are comprised of little more than circular shift registers, counters and comparators. These devices can be built with bounded fan in and fan out, and can operate in a systolic or (partially) self timed manner, without the benefit of global clock or control lines.

For a VLSI circuit model, the techniques show that if the inputs are read once, then a when- and where-determinate circuit that sorts n integers in the range $[0,m-1]$, for $n\le m\le n^{O(1)}$, must satisfy $AT^2=\Omega(n^2\log^2\frac{2m}{n})$. Moreover, this bound is the best possible for

the proof technique used.

It is worth observing that the proof of the AT^2 bound is based on the inevitable misallocation of the more significant input bits. The distribution of less significant input bits is irrelevant. In this sense, the standard proof for $k=1+\log n$ is somewhat misleading, since it draws attention to the locations of the least significant inputs.

This misallocation of more significant bits shows a little more about AT^2 bounds. In many applications, records are sorted according to a key field. We have shown that if the entire record is to be sorted according to a read-once, when- and where- determinate format, then the information content of the data field affects the AT^2 bound. For example, suppose that n records are to be sorted, that each record has a $\log n$-bit sort key, and that each record has a data field comprising $\Theta(\log n)$ bits of arbitrary information. Then $AT^2 = \Omega(n^2 \log^2 n)$. Circuits cannot beat this bound unless the data field is short, or the data is not arbitrary. Analogous remarks, evidently, hold for sorting circuits of minimum area.

Acknowledgements

I would like to thank Z. Kedem, F. T. Leighton, H. Trickey, J. D. Ullman and M.Y. Vardi for stimulating discussions and many helpful suggestions.

References

[AHU] Aho, A. V., J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, Mass., 1974.

[BP] Bilardi, G. and F. Preparata, "A Minimum Area VLSI Network for O(log N) Time Sorting," *Proc. Sixteenth Annual ACM Symposium on the Theory of Computing*, pp. 64-70, 1984.

[CJ] Courant, R., and F. John, *Introduction to Calculus and Analysis, Vol. I*, Interscience Publishers, 1965.

[CS] Cole, R., and A.R. Siegel, "Optimal VLSI Circuits for Sorting," Courant Institute Technical Report, 1985.

[F] Feller, W., *An Introduction to Probability and its Applications, Vol. II*, John Wiley & Sons, 1968.

[Ke] Kedem, Z.M., "Optimal Allocation of Area for Single-Chip Computations," *SIAM J. Computing* (to appear).

[Kn1] Knuth, D.E., *The Art of Computer Programming, Vol. I: Fundamental Algorithms*, Addison Wesley, Reading Mass., 1968.

[Kn3] Knuth, D.E., *The Art of Computer Programming, Vol. III: Sorting and Searching*, Addison Wesley, Reading Mass., 1973.

[KZ] Kedem, Z.M. and A. Zorat, "On Relations Between Input and Communication/Computation in VLSI," *Proc. 22nd Annual FOCS*, pp. 37-44, 1981.

[L] Leighton, F.T., "Tight Bounds on the Complexity of Parallel Sorting," *Proc. Sixteenth Annual ACM Symposium on the Theory of Computing*, pp. 71-80, 1984.

[S1] Siegel, A.R., "Minimum Storage Sorting Networks," *IEEE Trans. on Computers*, Special Issue on Sorting, April, pp. 355-361, 1985.

[S2] Siegel, A.R., "Computing information flow for lower bounds in VLSI circuits," Courant Institute technical report.

[T] Thompson, C.D., "Area-time complexity for VLSI," *Proc. Eleventh Annual ACM Symposium on the Theory of Computing*, pp. 81-88, 1979.

[U] Ullman, J. D., *Computational Aspects Of VLSI*, Computer Science Press, Rockville, Md., 1984.

This book may be kept

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

GAYLORD 142

PRINTED IN U S A

NYU COMP SCI TR-122 C 1
Siegel, Alan
Tight area bounds and
provably good AT{sup}2
bounds... Revised

NYU COMP SCI TR-122 C 1
Siegel, Alan
Tight area bounds and
provably good AT{sup}2
bounds... Revised

DATE DUE	BORROWER'S NAME

LIBRARY
N.Y.U. Courant Institute of
Mathematical Sciences
251 Mercer St.
New York, N. Y. 10012